DEPARTMENT OF TECHNICAL EDUCATION

D.Bannumaiah’s Educational Institution, Mysore

**DHARMAPRAKASHA**

**D.BANUMAIAH’S POLYTECHNIC**

***M.G.ROAD, UDAYAGIRI, MYSORE-570019***





**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

LAB RECORD

2024-25 (ODD SEM)

IN

FULL STACK DEVELOPMENT (20CS52I)

FOR

FIFTH SEMESTER COMPUTER SCIENCE & ENGINEERING

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name of the Student |  | | | | | | | | | |
| Register Number | ***3*** | ***2*** | ***5*** | ***C*** | ***S*** | ***2*** | ***2*** | ***0*** | **3** | **4** |

***COURSE COORDINATOR:*** *MALLIKARJUNA G N*

*SELECTION GRADE LECTURER*

Signature of the student: ………………………………..……..

***Course Co-ordinator Program Co-ordinator***

**LAB RECORD 2024-25.**

**D.BANUMAIAH’S POLYTECHNIC.**

**COMPUTER SCIENCE & ENGINEERING DEPT.**

|  |  |
| --- | --- |
| SL N0 | **Program** |
| 1 | How to create project plan and product backlog for project and User story creation. |
| 2 | Create sprint1 with required user stories |
| 3 | Create a wireframe for user stories. |
| 4 | Create a repository in GitHub and cloning the repository using VS code. |
| 5 | Create repository – named mini project-1 Push and pull operation in GitHub. |
| 6 | Create a form like registration form or feedback form, after submit hide create form and enable the display section using java script. |
| 7 | Create form validation using JavaScript |
| 8 | Create and run simple program in TypeScript |
| 9 | Forms - Use of HTML tags in forms like select, input, file, text area, etc. |
| 10 | Testing single page application (Registration form) using React. |
| 11 | Implement navigation using react router |
| 12 | Build single page application (Add Product to Product List) |
| 13 | Create Spring application with Spring Initializer using dependencies like  Spring web ,spring data jpa |
| 14 | Create REST controller for CRUD operations |
| 15 | Test created APIs with the help of postman |
| 16 | Writing Junit test cases for CRUD operations |
| 17 | Perform CRUD operation on MongoDB through RESST API using Spring Boot Starter Data |
| 18 | CRUD Operations on document using Mongo DB |
| 19 | Securing REST APIs with Spring Security |
| 20 | Build simple page application like shopping cart using ReactJS. |

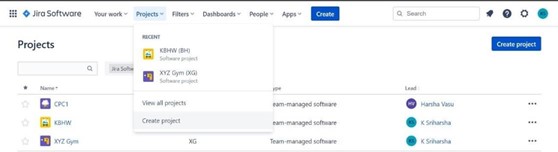
**Signature of Co-Ordinator**

**1.How to create project plan and product backlog for project and User story creation**.

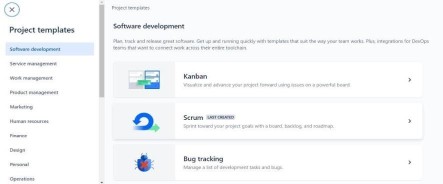
* 1. **Steps to create project in Jira**

Step 1 : Login into Atlassian Jira account.

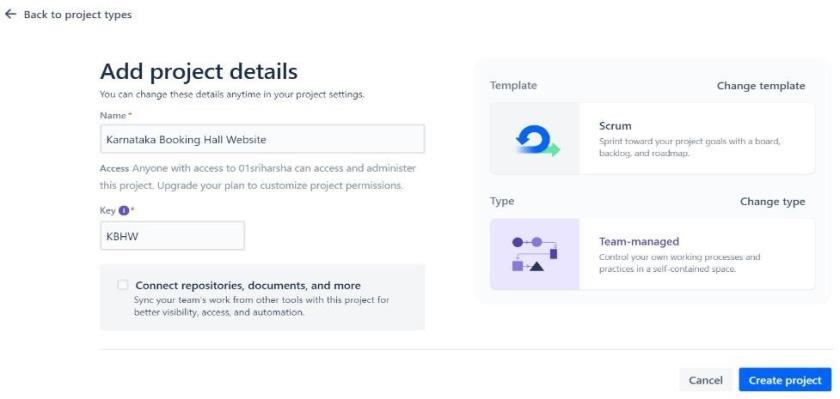
Step 2 : On the Jira software dashboard , Click on create project.



Step 3 : On next page , Select scrum project and then followed by team managed project



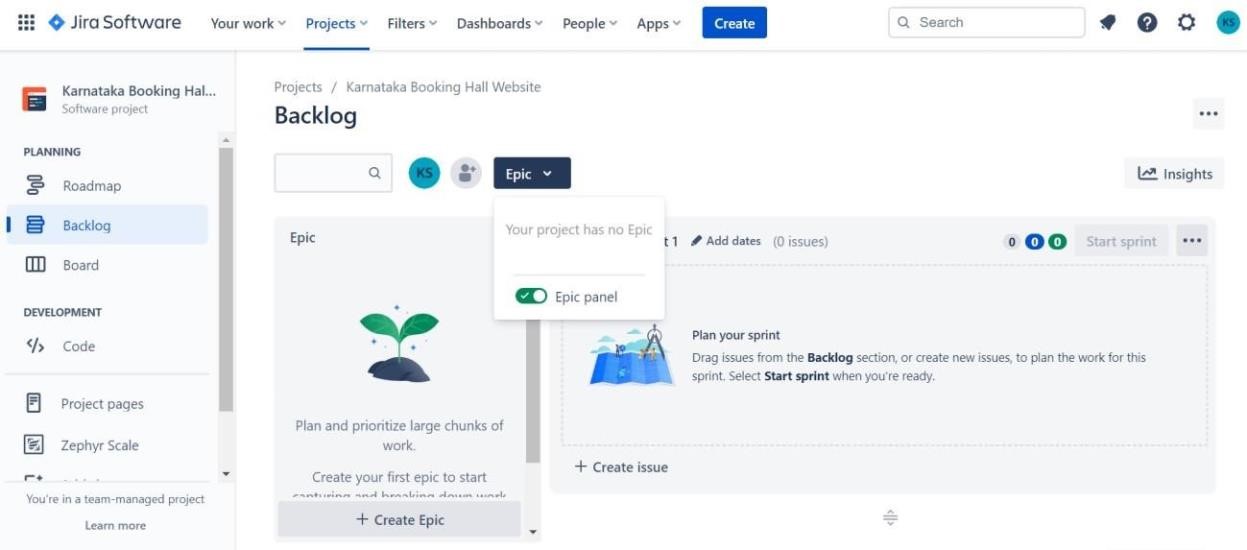
Step 4 : Enter project name and click on create project.



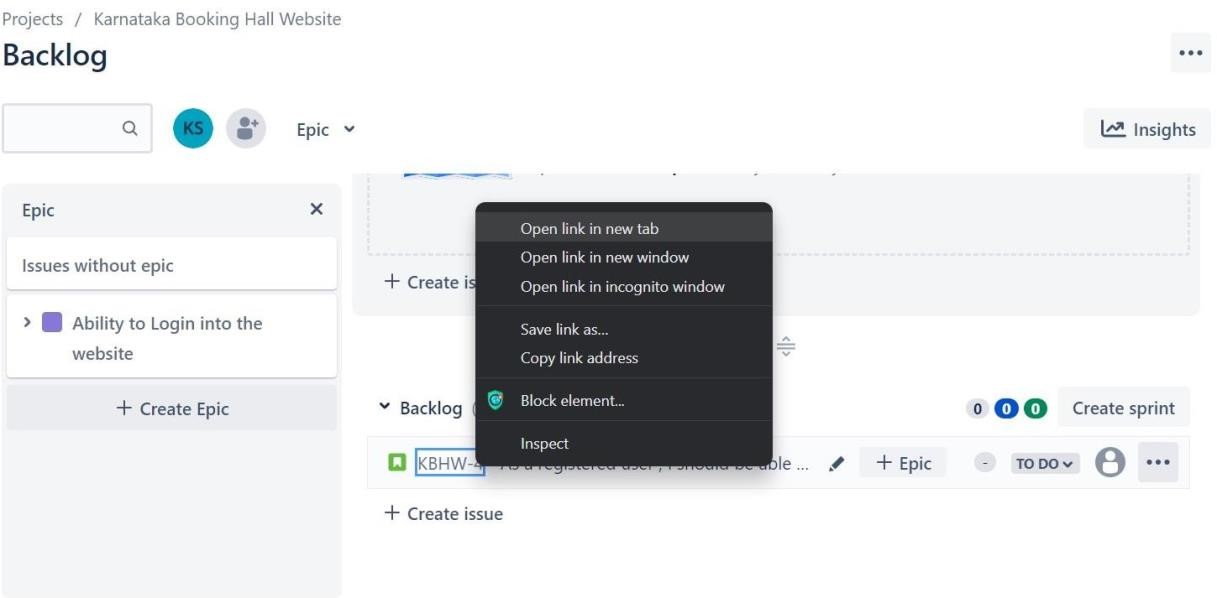
**Steps to Manage product backlog using Jira**

Step 1 : On the dashboard , Select backlog tab

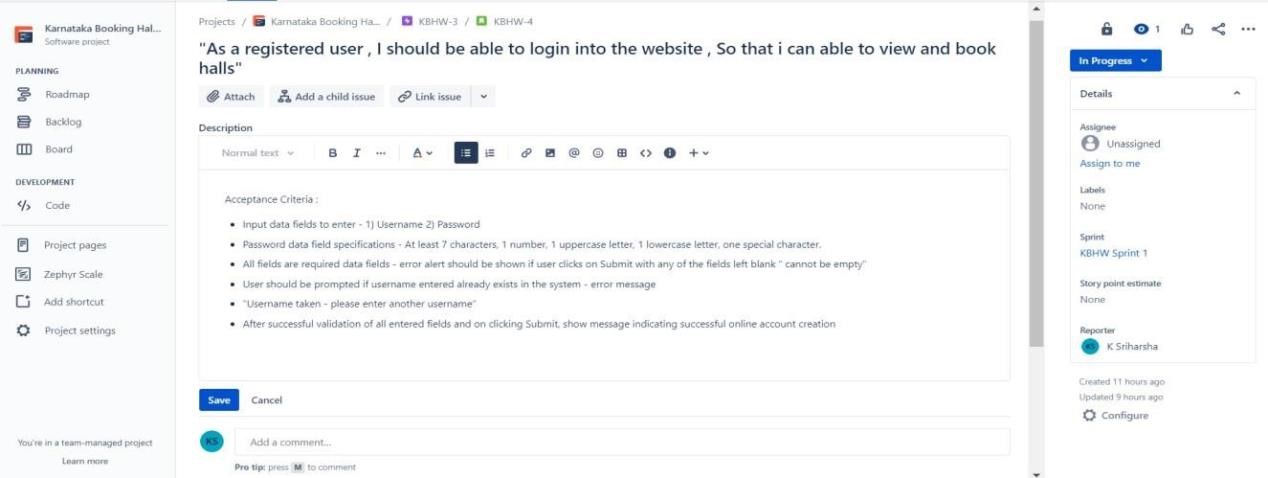
Step 2 : Select Epic option and toggle the epic switch to create a new Epic.



Step 3 : Add a new epic and then followed by user story by clicking on Create Issue option under backlog. Step 4 : After adding new issue right click on the issue id to open it in new tab.

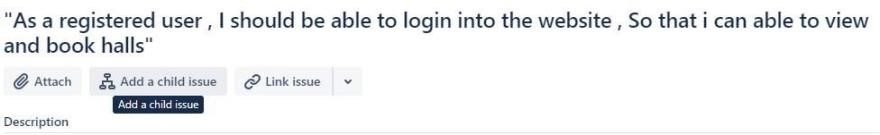


Step 5 : Under the description tab , add acceptance criteria for the specific issue.



Step 6 : After adding acceptance criteria , click on Add child issue

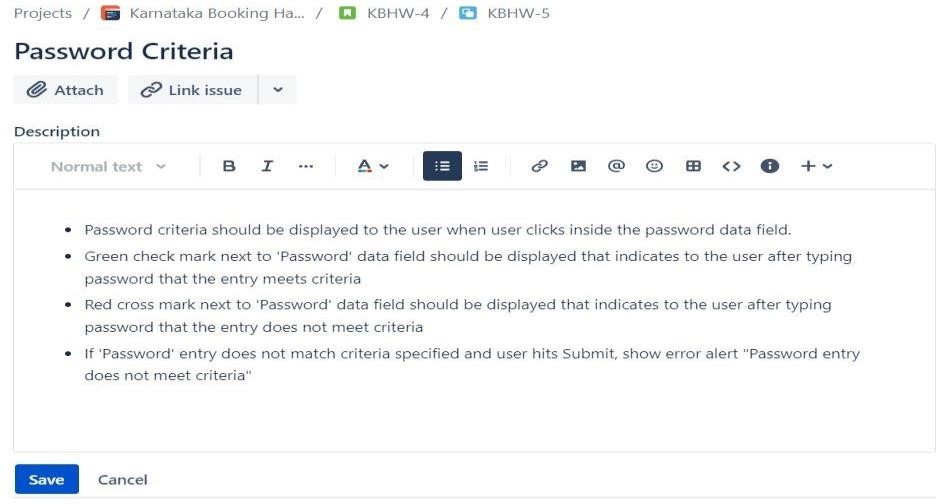
Step 6 : After adding acceptance criteria , click on Add child issue



Step 7 : Enter the name for the child issue and Click on it.



Step 8 : Enter the details to be done inside the description tab and click on save.

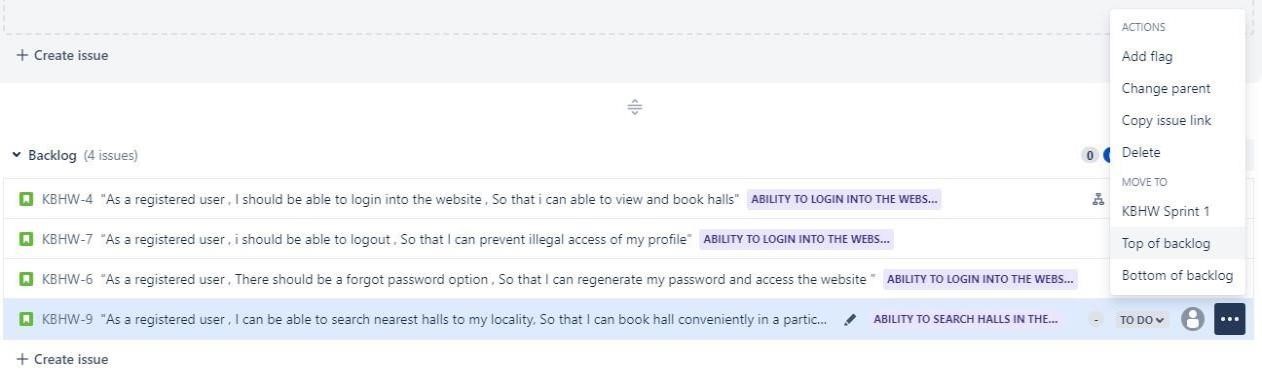


Step 9 : After creating the issue , head back to backlog dashboard and add Epic for the particular issue



Step 10 : Now prioritize the issue according to the requirements either by dragging it to top to bottom or by

selecting move option



**2. Create sprint1 with required user stories**

**Step 1 : Login into Atlassian Jira account.**

**Step 2 : On the Jira software dashboard , Click on create project and create a new scrum project**

**Step 3 : On the dashboard , Select backlog tab.**

**Step 4 : Select Epic option and toggle the epic switch to create a new Epic.**

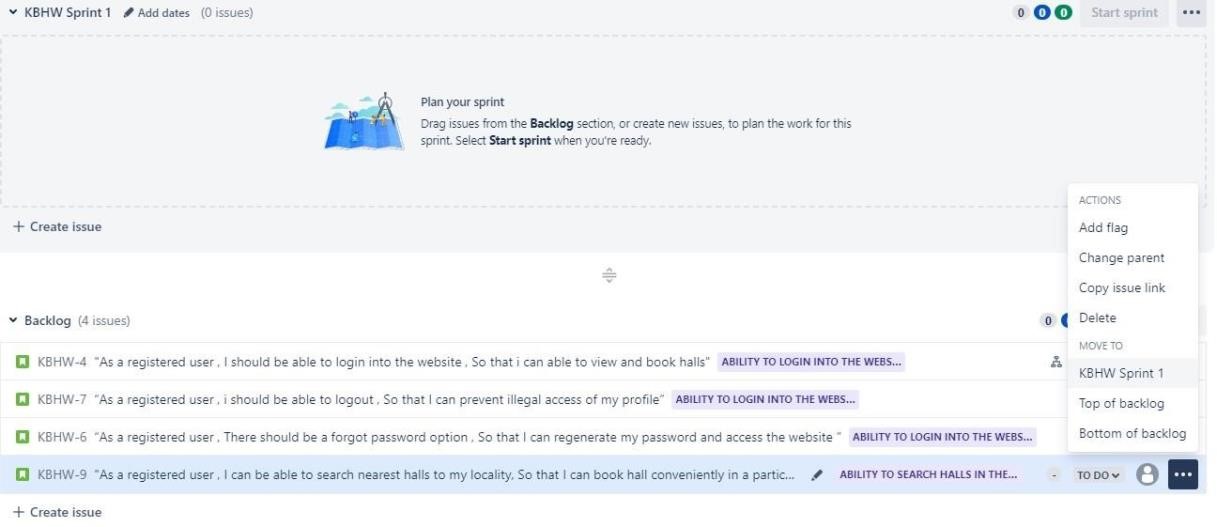
**Step 5 : Add a new epic and then followed by user story and child issue by clicking on Create Issue option under backlog**

**Step 6 : After creating the issue , add Epic for the particular issue.**



**Step 7 : Now drag and drop the issues from backlog to Sprint tab , which is above the backlog tab.**

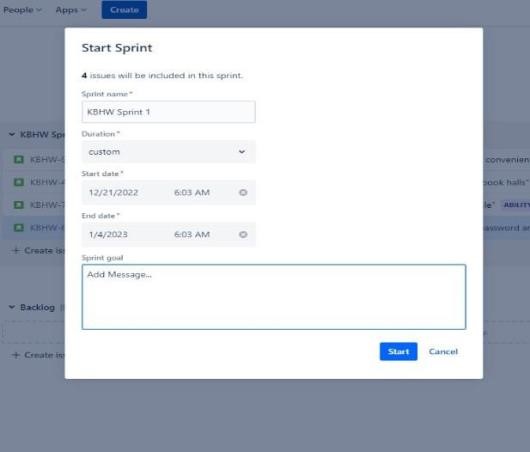
**Step 8 : Or select an issue click on three dot menu and select Move to Sprint .**



**Step 9 : After moving the issues from backlog to sprint , Click on start sprint button.**

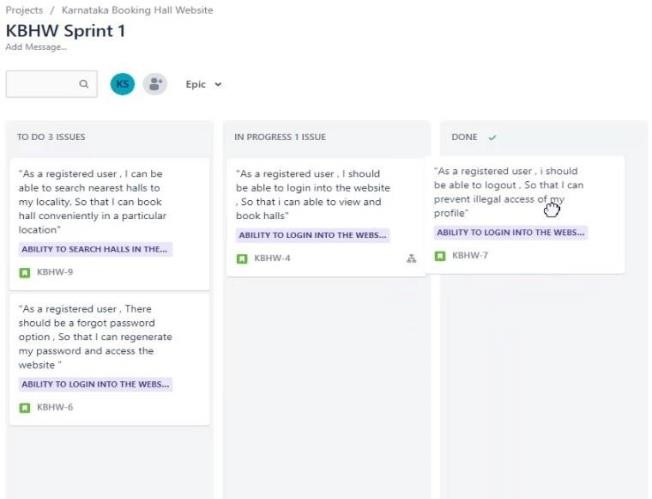


**Step 10 : A dialogue will open asking for the sprint duration , set the duration accordingly and click on start.**



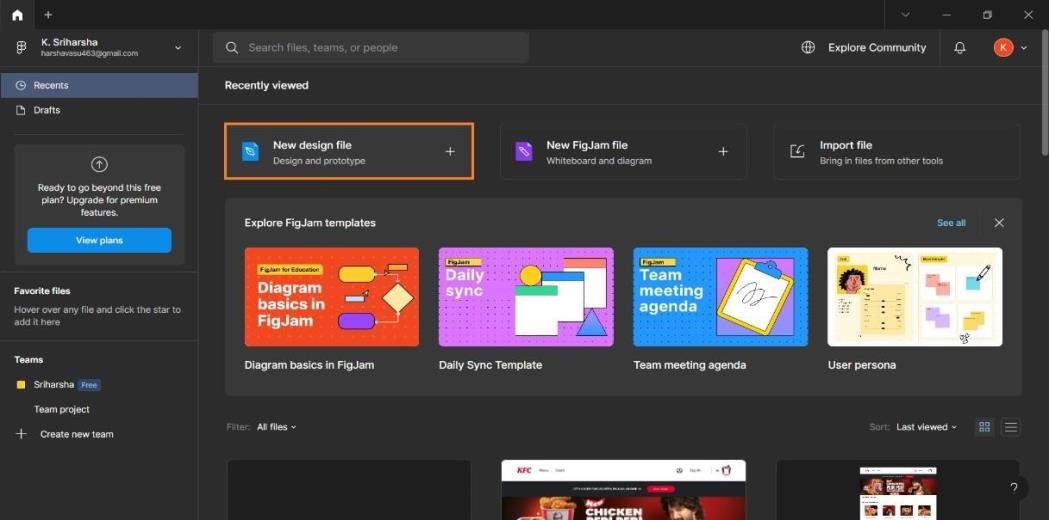
**Step 11 : Under the Board tab , we can see the issues that are in the To Do menu.**

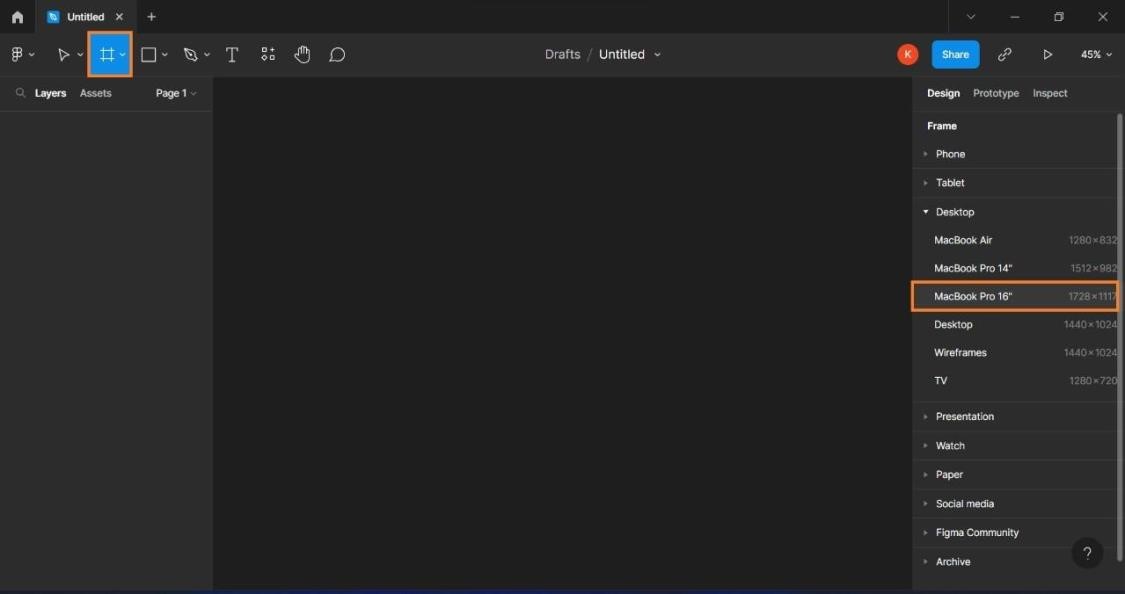
**Step 12 : According to the status of each issue , drag and drop the issue from To Do menu to In Progress or Done menu.**



**3. Create a wireframe for user stories**

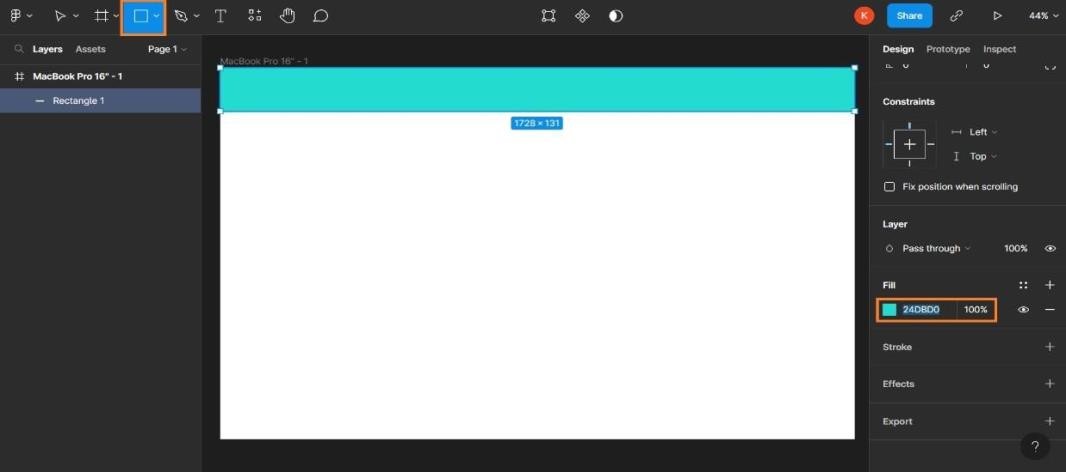
Step 1 : Login into figma website and downloard the figma desktop app

Step 2 : In the figma dashboard , Select New design file

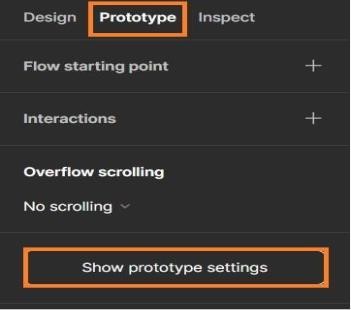
Step 3 : A blank workspace will open , Select the Frame and Reference device.

Step 4 : On the toolbar at the top , Select rectangle and start to design the UI of the web page.

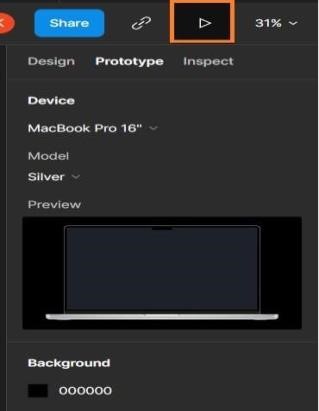
Step 5 : Use the necessary tools like line , text-box , hand tool and colour properties from the right side Design Tab.



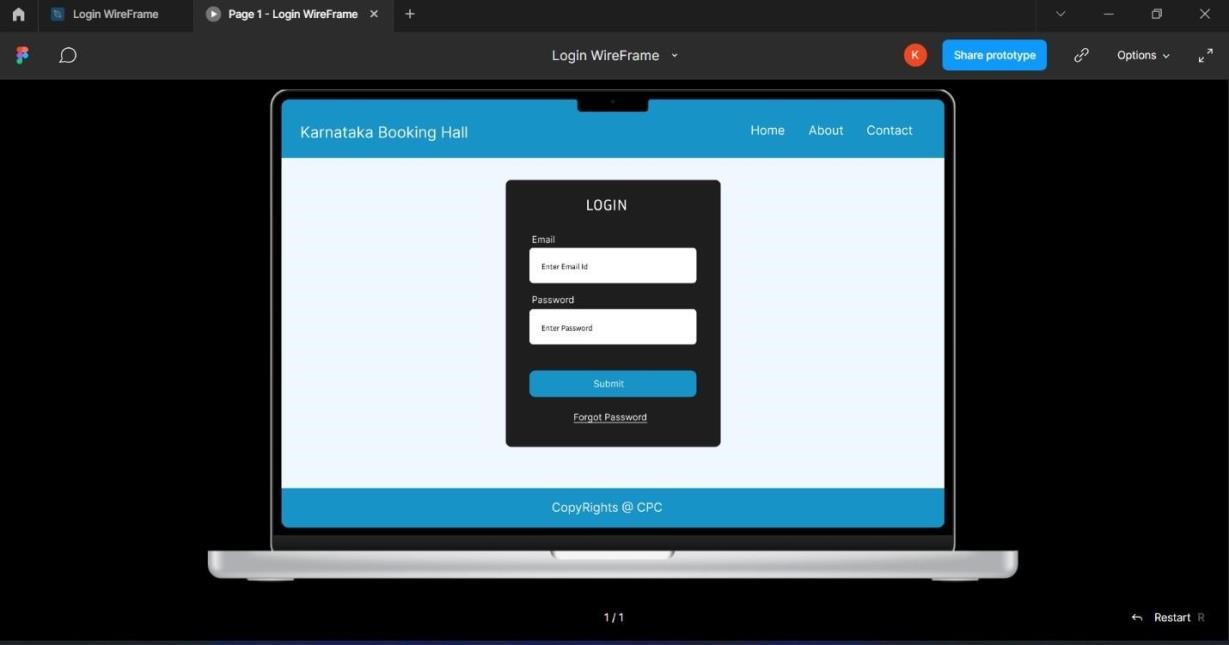
Step 6 : After building the Design , on the right menu Select Prototype menu and Click on show prototype settings.



Step 7 : Select the device to play, the model of the device and the background the prototype environment then click on play button on the top.



Step 8 : Now we can see how our designed UI will look in actual device.



**4.Create a repository in GitHub and cloning the repository using VS code.**

4.1 Creating an empty repository in Github through VS Code.

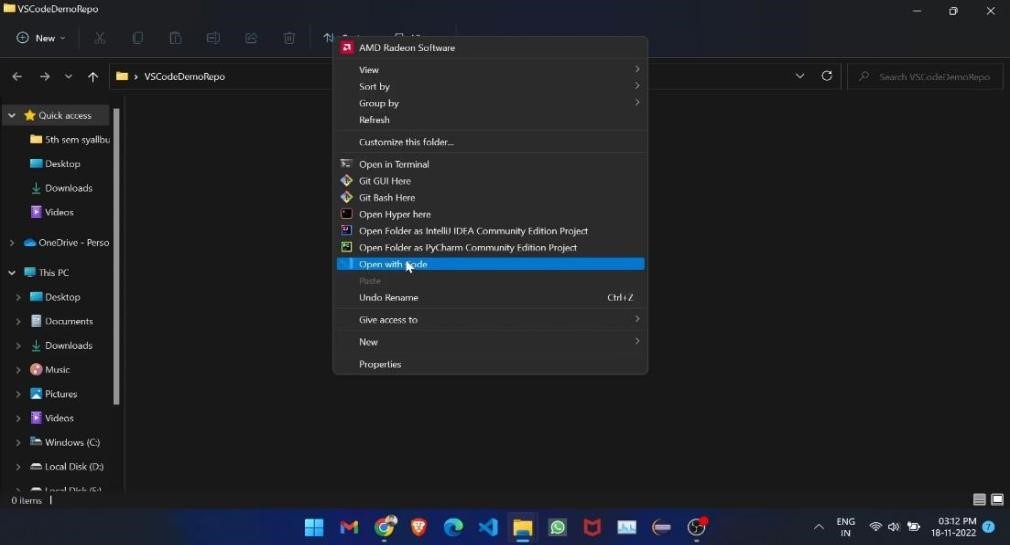
Step 1 : Login into github with the credentials.

Step 2 : Install VS Code editor to your desktop.

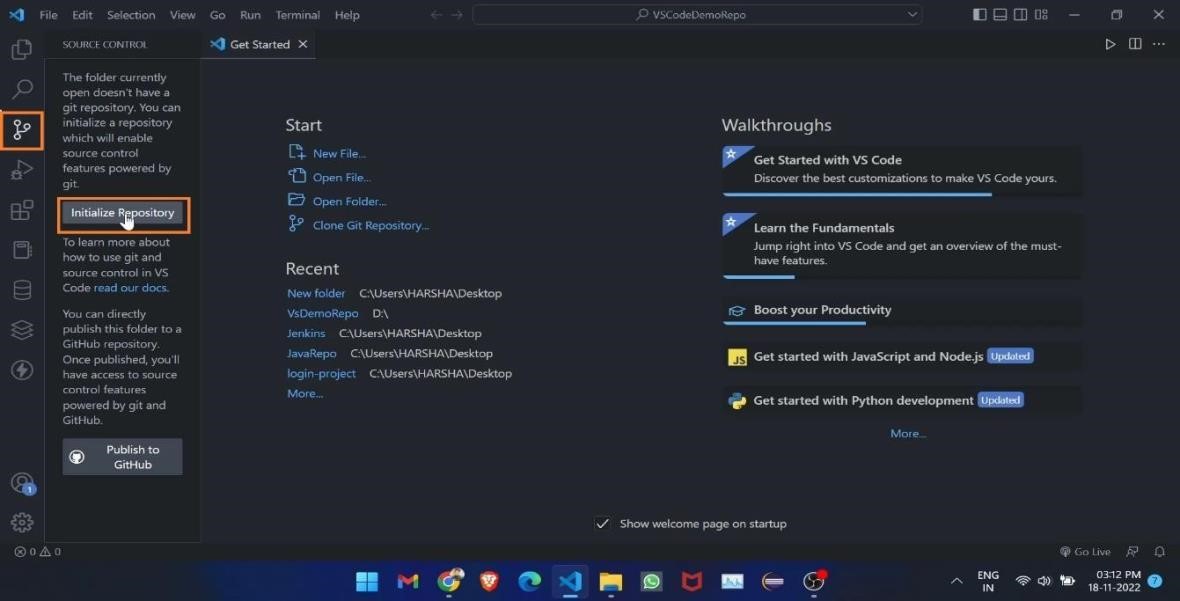
Step 3 : In the desktop , create a new empty folder by right click > new > folder



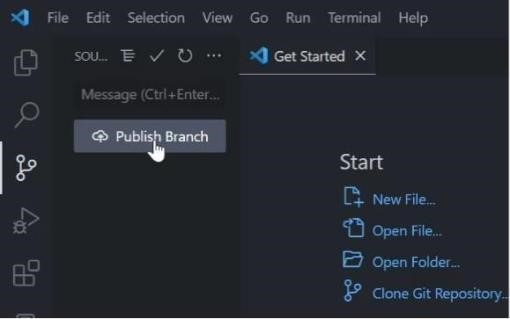
Step 4 : Open the folder and right click open with code.



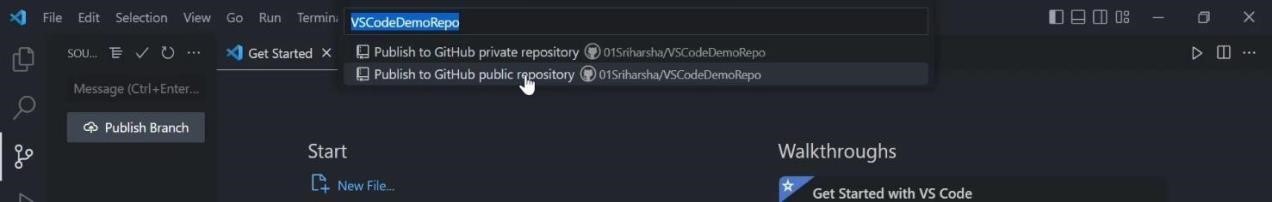
Step 5 : VS Code will be opened with the selected folder . On the left menu bar , Select Git icon and then select initialize repository option



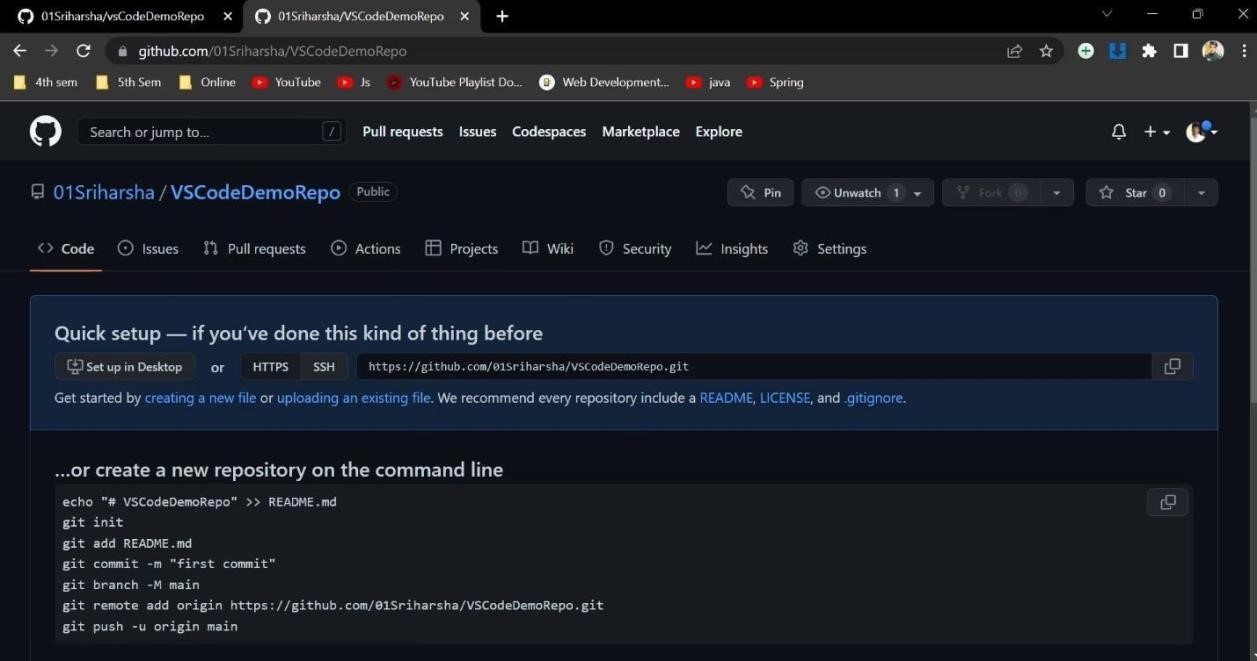
Step 6 : On next page , Click on Publish Branch . This will create a default Main branch.



Step 7 : A pop up window will open , Select Publish to public repository



Step 8 : Now head back to github, A new empty re1pository will be created with the folder name that was given at the beginning



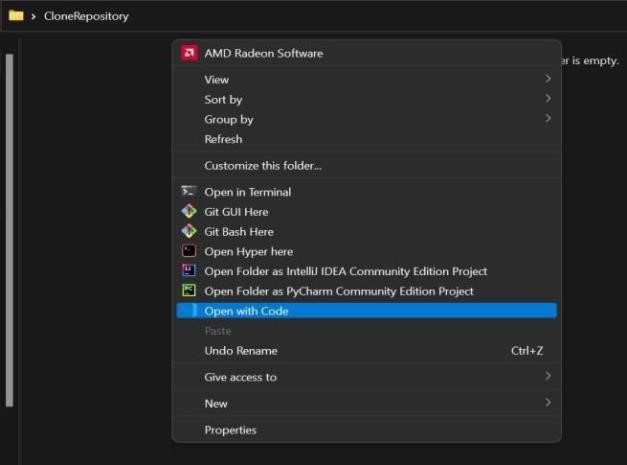
**4.2 Steps to Clone a github repository in VS Code**

Step 1 : Step 1 : Login into github with the credentials.

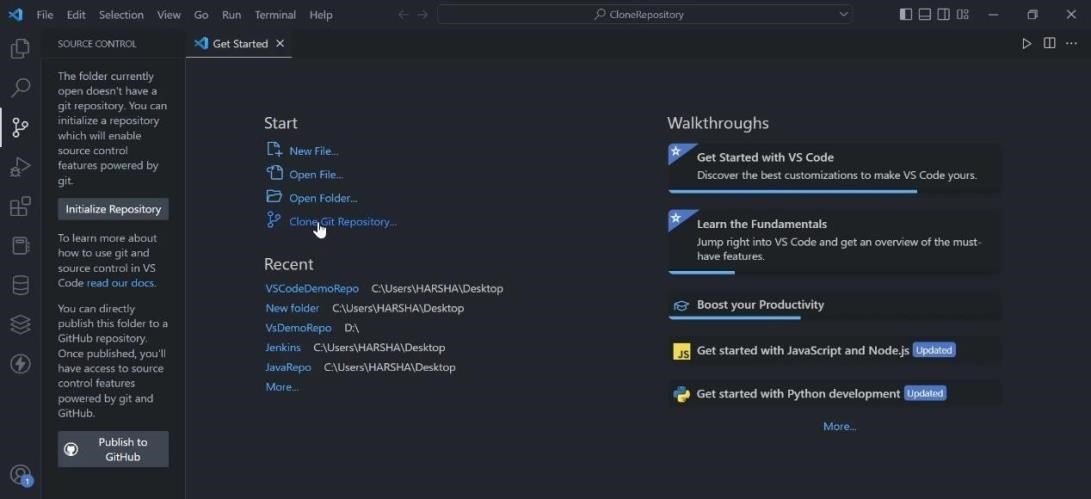
Step 2 : Install VS Code editor to your desktop.

Step 3 : In the desktop , create a new empty folder by right click > new > folder.

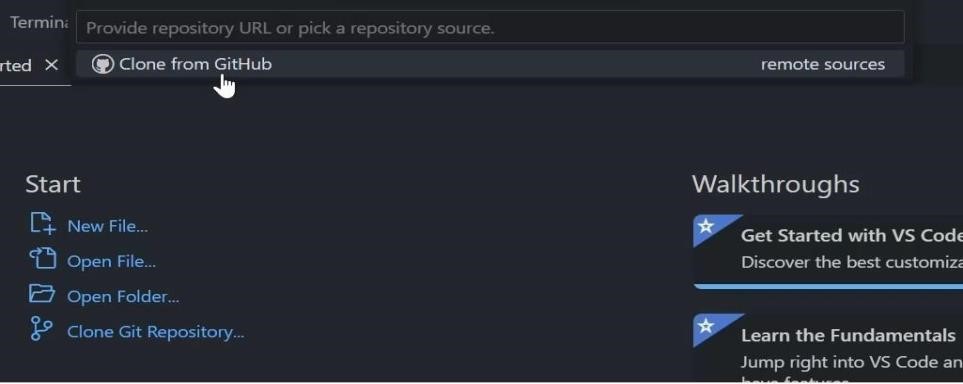
Step 4 : Open the folder and right click > open with code.



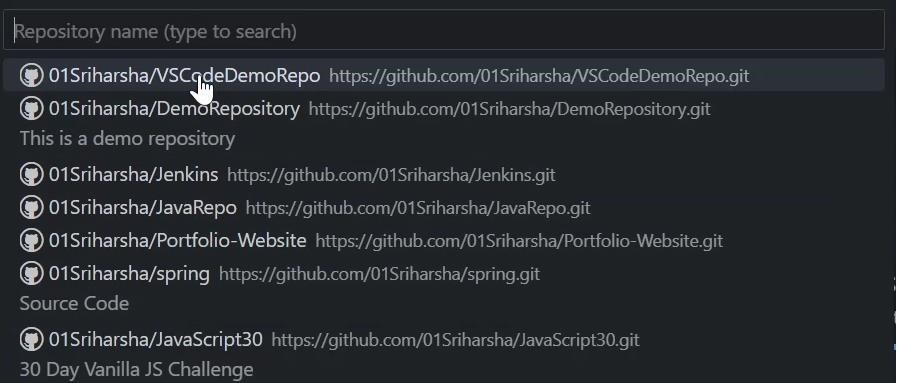
Step 5 : VS Code will be opened with the selected folder . Click on clone repository option.



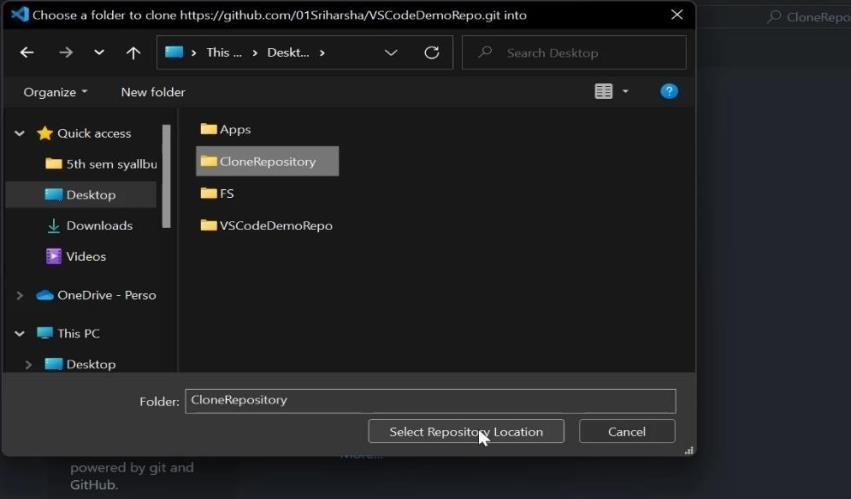
Step 6 : On the pop up window , Select clone from github . It will fetch all the repositories that are available globally and locally



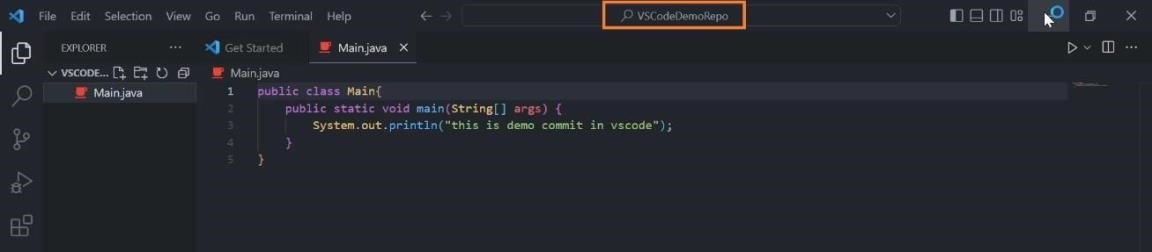
Step 7 : Search the repository you want to clone and click on it.



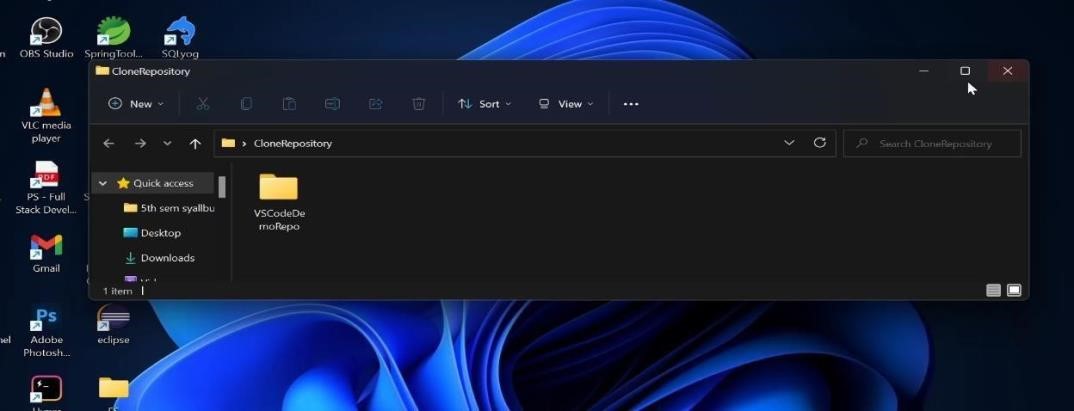
Step 8 : A dialogue will open , Select the folder where you want to clone the repository.



Step 9 : By clicking on select repository location , It will automatically open the cloned repository in VS Code.



Step 10 : The cloned repository will be stored in the folder that e created at the beginning.



**5. Perform the push and commit operation of project in Github through VS Code.**

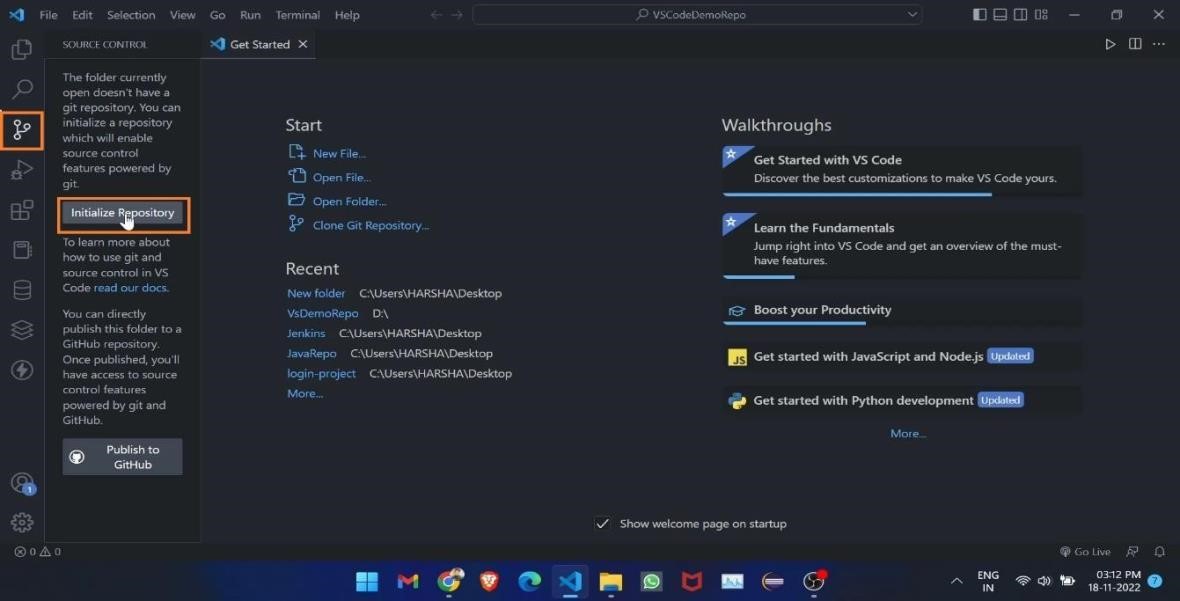
Step 1 : Login into github with the credentials.

Step 2 : Install VS Code editor to your desktop.

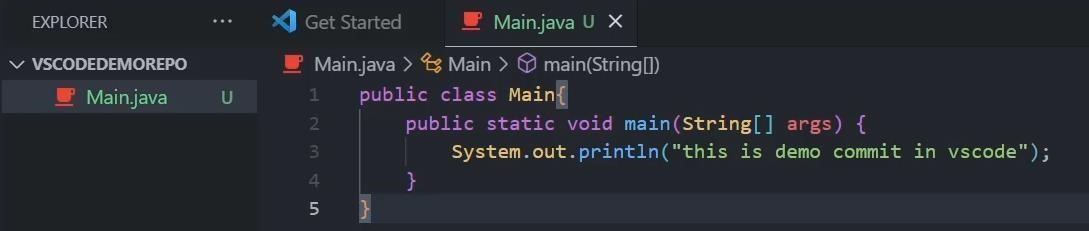
Step 3 : In the desktop , create a new empty folder by right click > new > folder.

Step 4 : Open the folder and right click open with code.

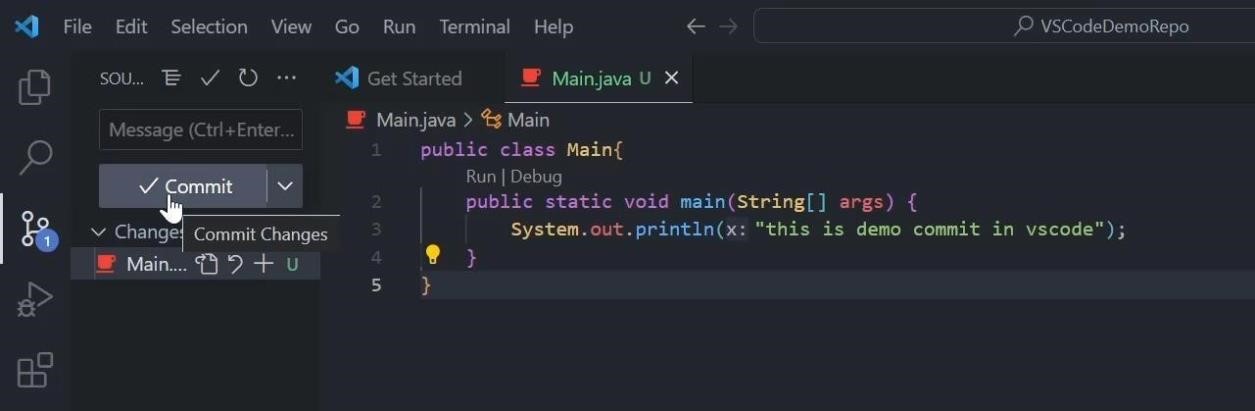
Step 5 : VS Code will be opened with the selected folder . On the left menu bar, Select Git icon and then select initialize repository option.



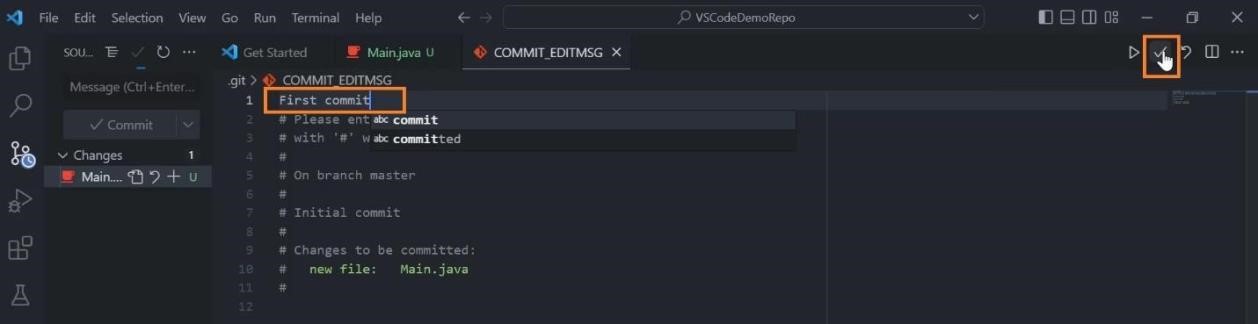
Step 6 : After initialization of the repository , Create a new file



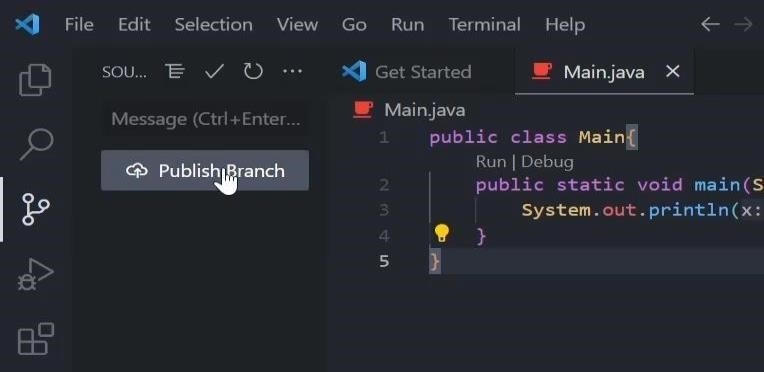
Step 7 : Select git menu on left , where we could see the modified or untracked file. Click on commit



Step 8 : Now enter the commit message and click on check mark on the right



Step 9 : Now click on publish branch (for new project) or Sync Changes (for existing project) to push the code into the github repository



Step 10 : The push will be reflected on the github with the commit message.

**6. Create a form like registration form or feedback form, after submit hide create form and enable the display section using java script.**

**Step 1: Create the HTML File**

1. Create a file named Registration.html.

2. Add the basic HTML structure and form.

<!DOCTYPE html>

<html>

<head>

<title>Registration Form</title>

<style> #details {

display: none; /\* Hide details section initially \*/

}

</style>

</head>

<body>

<h1>Registration Form</h1>

<!-- Registration Form -->

<div id="registrationForm">

<form onsubmit="return passValues(event);">

<label>Name:</label>

<input type="text" id="name" required /><br><br>

<label>Email:</label>

<input type="email" id="email" required /><br><br>

<label>Address:</label>

<input type="text" id="address" required /><br><br>

<input type="submit" value="Submit" />

</form>

</div>

<!-- Details Section -->

<div id="details">

<h2>Your Details</h2>

<p>Your Name: <span id="detailName"></span></p>

<p>Your Email: <span id="detailEmail"></span></p>

<p>Your Address: <span id="detailAddress"></span></p> </div>

<script>

function passValues(event) {

event.preventDefault(); // Prevent form submission

// Get values from the form

var name = document.getElementById("name").value; var email = document.getElementById("email").value; var address = document.getElementById("address").value;

// Store values in local storage localStorage.setItem("name", name); localStorage.setItem("email", email); localStorage.setItem("address", address);

// Hide the form and display the details

document.getElementById("registrationForm").style.display = "none"; document.getElementById("details").style.display = "block";

// Show the stored details

document.getElementById("detailName").innerText = name; document.getElementById("detailEmail").innerText = email; document.getElementById("detailAddress").innerText = address;

}

</script>

</body>

</html>

**Explanation**

**1. HTML Structure**: The form has three input fields (Name, Email, Address) and a submit button.

**2. CSS**: The details section is hidden by default using display: none;.

**3. JavaScript Function**:

o The passValues function is called when the form is submitted.

o It prevents the default form submission, retrieves the input values, and stores them in local storage.

o The registration form is hidden, and the details section is displayed with the entered values.

**Step 2: Test It**

1. Open Registration.html in a web browser.

2. Fill out the form and click "Submit."

3. You should see the form disappear and your details displayed below

**7. Create form validation using JavaScript**

**Step 1: Create index.html**

1. **Create a new file named index.html.** 2. **Add the following HTML structure:**

<!DOCTYPE html>

<html>

<head>

<title>Form Validation</title>

<script>

function validateForm() {

var name = document.myform.name.value; var password = document.myform.password.value;

// Check if name is empty if (name == null || name == "") { alert("Name can't be blank");

return false; // Prevent form submission

}

// Check if password is at least 6 characters long else if (password.length < 6) {

alert("Password must be at least 6 characters long."); return false; // Prevent form submission

}

return true; // Allow form submission

}

</script>

</head>

<body>

<h1>Registration Form</h1>

<form name="myform" method="post" action="valid.html" onsubmit="return validateForm()">

Name: <input type="text" name="name"><br/><br/>

Password: <input type="password" name="password"><br/><br/>

<input type="submit" value="Register"> </form>

</body>

</html>

**Step 2: Create valid.html**

1. **Create another file named valid.html.**
2. **Add the following content:**

<!DOCTYPE html>

<html>

<head>

<title>Validation Successful</title>

</head>

<body>

<h1>Validation Successful</h1>

</body>

</html>

**Step 3: Explanation of index.html**

* **HTML Structure**:
  + The form contains two fields: Name and Password.
  + It uses the onsubmit attribute to call the validateForm function before submission.
* **JavaScript Function**: o The validateForm function retrieves the values of the name and password fields.
  + It checks if the name field is empty and alerts the user if it is. o It checks if the password is less than 6 characters long and alerts the user if it is.
  + If both checks pass, the form submits to valid.html.

**Step 4: Test the Form**

1. Open index.html in a web browser.
2. Try submitting the form with an empty name or a password shorter than 6 characters to see the validation messages.
3. Enter valid information and submit the form. You should be redirected to valid.html with the message "Validation Successful."

**8. Create simple hello world application using type script.**

Step 1 : Install TypeScript into the system npm install typescript --save-dev

Step 2 : Check the TypeScript compiler version i.e tsc --version

Step 3 : Create a Html file , Index.ts file and link the Index.js file using script tag assuming it is already existed.

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Hello World</title>  </head>  <body>      <button onclick="handleClick()">Click here</button>  <script src="index.js"></script> |

Step 4 : write a function that prints hello world in Index.ts file.

function handleClick(){

let message:string = "Hello World";

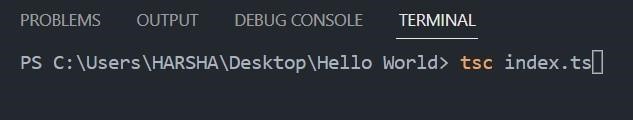
let root = document.createElement('h1');

root.textContent = message;

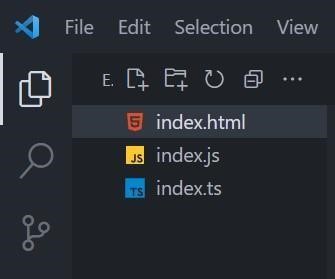
document.body.appendChild(root); }

Step 5 : Compile the Index.ts file by opening a new terminal

Step 6 : Type tsc index.ts and hit enter



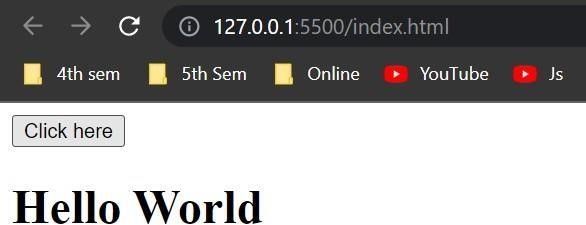
Step 7: ”Execution Policy - ExecutionPolicy RemoteSigned” enter this command in windows power shell/cmd



Step 8 : A new Javascript file will be created as Index.js.

Step 9 : Now run the Html file in the browser.

Step 10 : Click the button to display the hello world message.



**9. Forms - Use of HTML tags in forms like select, input, file, text area, etc.**

**Step 1: Create the HTML File**

a. Create a new file named simple\_form.html.

b. Add the basic HTML structure:

<!DOCTYPE html>

<html>

<head>

<title>Form Elements Example</title>

</head>

<body>

<h1>Form Elements Example</h1>

<form>

<!-- Text Box -->

<label for="name">Text Box:</label>

<input type="text" id="name" name="name" value="" /><br><br>

<!-- Radio Buttons -->

<label>Gender:</label><br>

<input type="radio" id="male" name="gender" value="male" />

<label for="male">Male</label><br>

<input type="radio" id="female" name="gender" value="female" /> <label for="female">Female</label><br><br>

<!-- Checkbox -->

<input type="checkbox" id="subscribe" name="subscribe" value="yes" /> <label for="subscribe">Subscribe to newsletter</label><br><br>

<!-- File Upload -->

<label for="file">Upload File:</label>

<input type="file" id="file" name="file" /><br><br>

<!-- Select Dropdown -->

<label for="sem">Semester:</label>

<select name="sem" id="sem">

<option value="1">1 Sem</option>

<option value="2">2 Sem</option>

<option value="3">3 Sem</option>

<option value="4">4 Sem</option>

</select><br><br>

<!-- Text Area -->

<label for="comments">Comments:</label><br>

<textarea id="comments" name="comments" rows="4" cols="50"></textarea><br><br>

<!-- Submit Button -->

<input type="submit" value="Submit" />

</form>

</body>

</html>

**Open form.html in a web browser.**

**Fill out the form and click "Submit" to see how it works.**

**10 Testing single page application (Registration form) using React.**

**Step 1: Set Up the Project**

1. Create a new React project (if you haven't already):

npx create-react-app registration-app cd registration-app

**Step 2: Create the Home Component**

1. Create a new file named Home.js in the src folder.

2. Add the following code to Home.js:

import { useState } from 'react'; import './App.css';

export default function Home() { // States for registration const [name, setName] = useState(''); const [email, setEmail] = useState(''); const [password, setPassword] = useState(''); const [submitted, setSubmitted] = useState(false);

const handleName = (e) => {

setName(e.target.value);

};

const handleEmail = (e) => { setEmail(e.target.value);

};

const handlePassword = (e) => { setPassword(e.target.value);

};

const handleSubmit = (e) => { e.preventDefault(); if (name === '' || email === '' || password === '') { alert("Please enter all the fields");

} else { setSubmitted(true);

}

};

// Showing success message const successMessage = () => { if (submitted) return (

<div className="success">

<h1>User {name} successfully registered!!</h1>

</div>

); };

return (

<div className="form">

<div>

<h1>User Registration</h1>

</div>

<div className="messages">

{successMessage()}

</div>

<form>

<fieldset>

<label className="label">Name</label>

<input onChange={handleName} className="input" value={name} type="text" /><br />

<label className="label">Email</label>

<input onChange={handleEmail} className="input" value={email} type="email" /><br />

<label className="label">Password</label>

<input onChange={handlePassword} className="input" value={password} type="password" /><br />

<button onClick={handleSubmit} className="btn" type="submit">

Submit

</button>

</fieldset>

</form>

</div>

);

}

**Step 3: Modify the Main Application File**

1. **Open src/index.js**.
2. **Update it to import and render the Home component**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import Home from './Home'; // Import Home component import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root')); root.render(

<React.StrictMode>

<Home /> {/\* Render Home component \*/}

</React.StrictMode>

);

reportWebVitals();

**Step 4: Add CSS Styles**

1. **Open src/App.css** (or create one if it doesn’t exist).
2. **Add the following styles:**

.input { width: 30%; padding: 12px 20px; margin: 8px 0; display: inline-block; border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box;

}

.label { display: block; margin: 10px 0 5px;

}

.success { color: green; margin: 10px 0;

}

**Step 5: Run Your Application**

1. **Start the application:** -npm start

**1. Open your browser and navigate to http://localhost:3000 to see the registration form. Step 6: Test the Registration Form**

1. **Try submitting the form without filling it out to see the alert.**
2. **Fill out the fields and submit to see the success message.**

**11 Implement navigation using react router**

**Step 1: Install React Router**

1. **Open your terminal in the root directory of your React application.**
2. **Run the following command to install React Router:**

-npm install react-router-dom

**Step 2: Set Up the Main Application File**

1. **Open src/index.js.**
2. **Replace the existing code with the following:** import ReactDOM from "react-dom/client";

import { BrowserRouter, Routes, Route } from "react-router-dom"; import Layout from "./pages/Layout"; import Home from "./pages/Home"; import Blogs from "./pages/Blogs"; import Contact from "./pages/Contact"; import NoPage from "./pages/NoPage";

const App = () => { return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Layout />}>

<Route index element={<Home />} />

<Route path="blogs" element={<Blogs />} />

<Route path="contact" element={<Contact />} />

<Route path="\*" element={<NoPage />} />

</Route>

</Routes>

</BrowserRouter>

);

};

const root = ReactDOM.createRoot(document.getElementById('root')); root.render(<App />);

**Step 3: Create the Pages Directory**

1. **Create a folder named pages in the src directory.**

**Step 4: Create Each Page Component**

1. **Create Home.js** 
   1. **Inside the pages folder, create a file named Home.js.**
   2. **Add the following code:** const Home = () => { return <h1>Home</h1>;

};

export default Home;

1. **Create Blogs.js** 
   1. **Create a file named Blogs.js in the pages folder.**
   2. **Add the following code:** const Blogs = () => { return <h1>Blog Articles</h1>;

};

export default Blogs;

* 1. **Create Contact.js**
  2. **Create a file named Contact.js in the pages folder.**
  3. **Add the following code:** const Contact = () => { return <h1>Contact Me</h1>;

};

export default Contact;

**4. Create NoPage.js**

1. **Create a file named NoPage.js in the pages folder.**
2. **Add the following code:**

const NoPage = () => { return <h1>404 - Page Not Found</h1>;

};

export default NoPage;

**Step 5: Create the Layout Component**

1. **Create a file named Layout.js in the pages folder.**
2. **Add the following code:** import { Outlet, Link } from "react-router-dom";

const Layout = () => { return ( <>

<nav>

<ul>

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/blogs">Blogs</Link>

</li>

<li>

<Link to="/contact">Contact</Link>

</li>

</ul>

</nav>

<Outlet />

</>

);

};

export default Layout;

**Step 6: Add CSS Styles**

1. **Open or create src/App.css.**
2. **Add the following CSS styles for navigation:**

ul { list-style-type: none; margin: 0; padding: 0; overflow: hidden; background-color: #04AA6D;

}

li { float: left; border-right: 1px solid #bbb;

}

li a {

display: block; color: white; text-align: center; padding: 14px 16px; text-decoration: none;

}

li a:hover:not(.active) {

background-color: #111;

}

**Step 7: Run Your Application**

1. **Start the application:** - npm start
2. **Open your browser and navigate to** [**http://localhost:3000.**](http://localhost:3000/)

**Step 8: Test Navigation**

1. Click on the "Home," "Blogs," and "Contact" links to navigate between pages.
2. Try entering a URL that doesn’t exist (like /random) to see the 404 page.

**12. Build single page application (Add Product to Product List)**

**App.js**

**Step 1: Set Up Your Project**

1. **Create a React App**: If you haven't already, you can create a new React application **using Create React App. Run this command in your terminal**:

-npx create-react-app product-list

**Navigate into your project folder:**

-cd product-list

**Open the Project**: Open the project in your preferred code editor.

**Step 2: Modify App.js**

* 1. **Open src/App.js**: This file is where you'll build the main component of your app.
  2. **Import Necessary Hooks**: At the top of the file, import the useState hook from React.

-import { useState } from "react";

* 1. **Create the App Function**: Define your main App component using the function syntax. function App() {

// Your state variables will go here

}

}export default App;

* 1. **Set Up State Variables: Use the useState hook to create two state variables: one for the product list and another for the input value.**

**const [list, setList] = useState([]); const [value, setValue] = useState("");**

* 1. **Add Function to Handle Adding Products:**
* **Create a function called addToList that adds the product to the list.**
* **Ensure that the input is not empty.**

**const addToList = () => { if (value.trim() === "") return; // Prevent adding empty products setList((prevList) => [...prevList, value]);**

* 1. **Add Function to Handle Deleting Products:**
* **Create a function called deleteItem that removes a product from the list by its index.**

**const deleteItem = (index) => { setList((prevList) => prevList.filter((\_, i) => i !== index)); };**

* 1. **Build the UI: In the return statement, create a simple form to input products and display the product list.**

**return (**

**<div className="App">**

**<fieldset>**

**<h2>Add Product to List</h2>**

**<input type="text" value={value} onChange={(e) => setValue(e.target.value)} placeholder="Enter product name"**

**/>**

**<button onClick={addToList}>Click to Add</button>**

**<br /><br />**

**<h2>Product Catalog</h2>**

**<ol>**

**{list.map((item, i) => (**

**<li key={i} onClick={() => deleteItem(i)}>{item}</li>**

**))}**

**</ol>**

**<h3>Click on Product to Delete</h3>**

**</fieldset>**

**</div>**

**);**

**8 Export the App Component: At the bottom of the file, make sure to export your export default App;**

**Step 3: Modify index.js**

1. **Open src/index.js**: This is where your app is rendered into the DOM.
2. **Import the App Component**: Make sure you have the import for your App component.

import App from './App';

1. **Render the App Component**: The existing code should already render your App component correctly.

If needed, ensure it looks like this:

const root = ReactDOM.createRoot(document.getElementById('root')); root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

**Step 4: Add Basic Styles (Optional)**

1. **Open src/App.css**: You can add some basic styles for better visuals. Here’s an example: .App { max-width: 600px; margin: auto; padding: 20px; text-align: center;

}

input {

padding: 10px; width: 80%; margin-bottom: 10px;

}

button { padding: 10px 20px;

}

ol { list-style-type: none; padding: 0;

}

li { cursor: pointer; padding: 5px; background: #f0f0f0; margin: 5px 0;

}

li:hover { background: #e0e0e0;

}

**Step 5: Run Your Application**

1. **Start the Development Server**: In the terminal, run: npm start

This should open your application in the default web browser.

# 13. Create Spring application with Spring Initializer using dependencies like Spring Web, Spring Data JPA

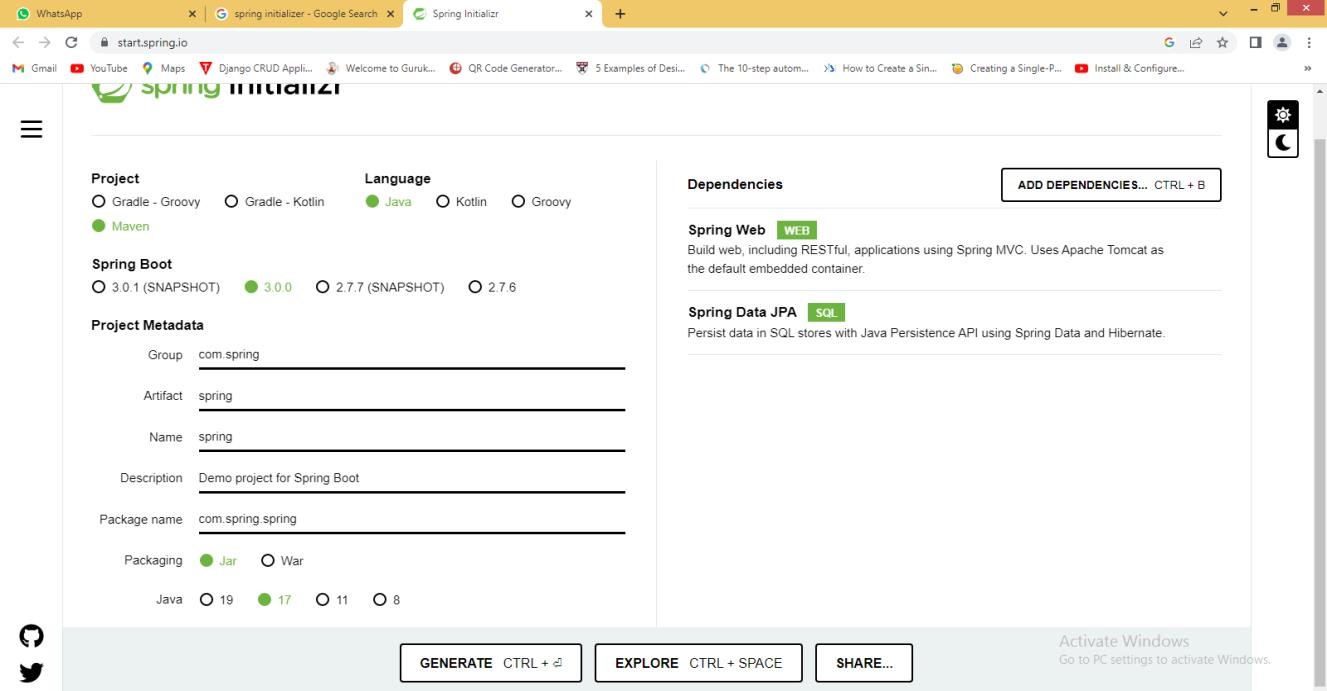
**Step 1: Access Spring Initializr**

1. **Open Your Browser**: o Go to [Spring Initializr.](https://start.spring.io/)

**Step 2: Configure Your Project**

1. **Choose Project Metadata**:
   * **Project**: Select **Maven Project**. o **Language**: Choose **Java**. o **Spring Boot Version**: Select the latest stable version (e.g., 3.x.x).
2. **Project Metadata**:
   * **Group**: com.example o **Artifact**: springbootapp o **Name**: springbootapp o **Description**: Demo project for Spring Boot o **Package Name**: com.example.springbootapp o **Packaging**: Select **Jar**.

**Java Version**: Choose your installed Java version (e.g., 17).



**3 Add Dependencies**:

o Click on **Add Dependencies** and select:

* + - **Spring Web**
    - **Spring Data JPA**
    - (Optional: Add H2 Database for in-memory database testing)

**Step 3: Generate and Download the Project**

1. **Generate the Project**:
   * Click the **Generate** button.
   * Download the zip file containing your project.

**Step 4: Import the Project into Eclipse**

1. **Open Eclipse**:

* + Go to File → Import.

1. **Import Maven Project**:
   * Select Maven → Existing Maven Projects and click Next.
2. **Browse to the Project**:
   * Click Browse and locate the extracted zip file folder. o Select the folder and click OK.
3. **Finish the Import**: o Click Finish to import the project.

**Step 5: Modify the Main Application Class**

1. **Locate the Main Class**:

* + In src/main/java/com/example/springbootapp, find the main application file (e.g., SpringbootappApplication.java).

**Add a Welcome Message**:

• Inside the main method, add the following line:

- System.out.println("Welcome to Spring Boot Application");

Here’s what it might look like:

package com.example.springbootapp; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication public class SpringbootappApplication { public static void main(String[] args) {

System.out.println("Welcome to Spring Boot Application");

SpringApplication.run(SpringbootappApplication.class, args);

}

}

**Step 6: Run the Application**

1. **Run as Spring Boot App**:

o Right-click on the main class (SpringbootappApplication.java). o Select Run As → Spring Boot App.

1. **Check Console Output**: o In the Eclipse console, you should see:

Welcome to Spring Boot Application

# Create REST controller for CRUD operations

**Step 1: Install STS4**

**1. Open Eclipse:**

o **Go to Help → Eclipse Marketplace.** o **Search for STS4 (Spring Tool Suite 4).** o **Click Go and install it.**

**Step 2: Create a New Spring Starter Project**

1. **Create Project:** 
   * **Click on File → New → Project.**
   * **Select Spring Starter Project and click Next.**
2. **Project Details:** 
   * **Name: Springboot-first-app.**
   * **Dependencies: Add:** 
     + **Spring Web**
     + **Spring Data JPA**
     + **MySQL Driver**
   * **Click Finish to create the project.**

**Step 3: Create Packages**

**1. Create Packages:**

o **In src/main/java/com/example/demo, create the following packages:**

* **entity**
* **controller**
* **repository**

**Step 4: Create User Class, Repository, and Controller**

**User.java (Entity)**

1. **Create User.java:** o **In the entity package, create a class named User.java.**
2. **Add Code: java Copy code package com.example.demo.entity;**

**import javax.persistence.\*;**

**@Entity**

**@Table(name="user") public class User {**

**@Id**

**@GeneratedValue(strategy=GenerationType.AUTO) private Long id; private String firstname; private String lastname; // Fixed typo from lasttname to lastname**

**// Default constructor public User() {}**

**// Parameterized constructor public User(String firstname, String lastname) { this.firstname = firstname; this.lastname = lastname;**

**}**

**// Getters and Setters public Long getId() { return id; } public void setId(Long id) { this.id = id; } public String getFirstname() { return firstname; } public void setFirstname(String firstname) { this.firstname = firstname; } public String getLastname() { return lastname; } // Fixed typo public void setLastname(String lastname) { this.lastname = lastname; } // Fixed typo**

**}**

**UserRepository.java (Repository)**

1. **Create UserRepository.java:**

o **In the repository package, create an interface named UserRepository.java.**

1. **Add Code: java Copy code package com.example.demo.repository;**

**import com.example.demo.entity.User; import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.stereotype.Repository;**

**@Repository**

**public interface UserRepository extends JpaRepository<User, Long> {} UserController.java (Controller)**

1. **Create UserController.java:**

o **In the controller package, create a class named UserController.java.**

1. **Add Code: java**

**Copy code**

**package com.example.demo.controller;**

**import com.example.demo.entity.User; import com.example.demo.repository.UserRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.\*;**

**import java.util.List;**

**@RestController**

**@RequestMapping("/users") public class UserController {**

**@Autowired private UserRepository userRepository;**

**@GetMapping public List<User> getAllUsers() { return this.userRepository.findAll();**

**}**

**@GetMapping("/{id}") public User getUserById(@PathVariable(value="id") long userId) { return this.userRepository.findById(userId).orElseThrow();**

**}**

**@PostMapping public User createUser(@RequestBody User user) { return this.userRepository.save(user);**

**}**

**@PutMapping("/{id}")**

**public User updateUser(@RequestBody User user, @PathVariable("id") long userId) {**

**User existingUser = this.userRepository.findById(userId).orElseThrow(); existingUser.setFirstname(user.getFirstname()); existingUser.setLastname(user.getLastname()); // Fixed typo return this.userRepository.save(existingUser);**

**}**

**@DeleteMapping("/{id}")**

**public ResponseEntity<User> deleteUser(@PathVariable("id") long userId) { User existingUser = this.userRepository.findById(userId).orElseThrow(); this.userRepository.delete(existingUser); return ResponseEntity.ok().build();**

**}**

**}**

**Step 5: Configure application.properties**

1. **Open application.properties:** o **Locate the application.properties file in src/main/resources.**
2. **Add Database Configuration:**

**properties Copy code spring.datasource.url=jdbc:mysql://localhost:3306/emp spring.datasource.username=root spring.datasource.password=root spring.jpa.hibernate.ddl-auto=update Step 6: Run Your Application**

**1. Run as Spring Boot App:**

o **Right-click on your main application class (e.g., SpringbootFirstAppApplication.java).** o **Select Run As → Spring Boot App.**

**15. Test created APIs with the help of Postman**

**Note: Create crud operation to Test with Postman**

**Step 1: Download & Install Postman**

1. **Visit Postman Website:** o **Go to** [**Postman Downloads.**](https://www.postman.com/downloads/)
2. **Download & Install:**

o **Download the installer for your operating system and follow the installation instructions. Step 2: Create a Collection in Postman**

1. **Open Postman.**
2. **Create a Collection:** 
   * **Click on the Collections tab on the left sidebar.** o **Click on + New Collection.**
   * **Give your collection a name (e.g., User API Collection) and click Create.**

**3. Add Requests:**

* + **Inside your new collection, click on Add Request.**

**Step 3: Demonstrate CRUD Operations**

**1. GET Method**

* **Select GET Method**: o In the request tab, select **GET** from the dropdown.
* **Enter the URL**:

o Input the URL: http://localhost:8080/users.

* **Send the Request**:

o Click on **Send**. o You should see a response with a list of users (if any exist).

**2. POST Method**

* **Select POST Method:** o **Click on Add Request again, and this time select POST.**
* **Enter the URL:** o **Input the URL: http://localhost:8080/users.**
* **Set Body:** 
  + - **Click on the Body tab.**
    - **Select raw and then choose JSON from the dropdown.**
* **Enter JSON Input:**

o **Input the following JSON (example):**

**{**

**"firstname": "John",**

**"lastname": "Doe"**

**}**

* **Send the Request:**

o **Click on Send.** o **You should see a response with the newly created user data.**

1. **PUT Method** 
   * **Select PUT Method:** o **Click on Add Request again, and select PUT.**
   * **Enter the URL:** 
     + **Input the URL (replace 1 with the actual user ID you want to update): http://localhost:8080/users/1.**
   * **Set Body:** 
     + **Click on the Body tab.**
     + **Select raw and then choose JSON from the dropdown.**
   * **Update JSON Input:** 
     + **Input the updated data in JSON format:**

**{**

**“firstname”:”Jane”,**

**“lastname”:”Doe”**

**}**

* + **Send the Request:** 
    - **Click on Send.** o **You should see a response with the updated user data.**

1. **DELETE Method** 
   * **Select DELETE Method:** 
     + **Click on Add Request again, and select DELETE.**
   * **Enter the URL:** 
     + **Input the URL (replace 1 with the actual user ID you want to delete): http://localhost:8080/users/1.**
   * **Send the Request:** 
     + **Click on Send.**

**You should see a response indicating the user has been deleted (often an empty response or confirmation message).**

**16 . Writing Junit test cases for CRUD operations**

**Note**: Create crud operation to Test with Junit

**Step 1:** Download JUnit

1. Visit the JUnit Website: o Go to JUnit 4 Downloads.

2. Download JAR Files: o Find the section for Plain-old Jar and download the following files:

▪ junit.jar

▪ hamcrest-core.jar

3.Create a Folder:

o Create a folder on your drive (e.g., C:\JUnit\) and copy both JAR files into this folder.

**Step 2:** Create a Project in Eclipse

1.Open Eclipse:

o Launch your Eclipse IDE.

2.Create New Project:

o Click on File → New → Java Project. o Enter the project name (e.g., SpringbootFirstApp).

**Step 3:** Configure Build Path

1.Right-Click on Project:

o In the Project Explorer, right-click on your project name.

2.Select Build Path:

o Click on Build Path → Configure Build Path.

3.Add External JARs:

o Go to the Libraries tab.

o Click on Add External JARs.

o Navigate to the folder where you saved the JAR files and select both junit.jar and hamcrestcore.jar.

o Click Open and then Apply and Close.

**Step 4:** Create Test Class

1.Locate Test Folder:

o In your project structure, navigate to src/test/java.

2.Create Test Class:

o Right-click on the default package (or create a new package for tests, e.g., com.example.demo).

o Select New → Class. FSD Dept. of CS&E, DDBP, Mysore 51

o Name the class SpringbootFirstAppApplicationTests.

**Step 5:** Write Test Cases

Open SpringbootFirstAppApplicationTests.java and add the following code:

**package com.example.demo; // Adjust the package name based on your structure**

**import static org.assertj.core.api.Assertions.\*; import static org.junit.Assert.assertNotNull; import static org.junit.Assert.assertNotEquals; import java.util.List; import org.junit.Test; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.context.SpringBootTest;**

**@SpringBootTest public class SpringbootFirstAppApplicationTests {**

**@Autowired**

**UserRepository userRepo;**

**@Test**

**public void testCreate() {**

**User u = new User();**

**u.setId(3L); // Ensure this ID is not conflicting with existing data**

**u.setFirstname("Kavya");**

**u.setLastname("Shree"); userRepo.save(u); assertNotNull(userRepo.findById(3L).get()); // Use the correct ID }**

**@Test public void testReadAll() {**

**List<User> list = userRepo.findAll(); assertThat(list).hasSizeGreaterThan(0); // Assert that list is not empty**

**}**

**@Test public void testUpdate() {**

**User u = userRepo.findById(2L).get(); // Ensure the ID exists**

**u.setFirstname("Murthy"); userRepo.save(u);**

**assertNotEquals("Niranjan", userRepo.findById(2L).get().getFirstname()); // Check if updated correctly**

**}**

**@Test public void testDelete() { userRepo.deleteById(2L); // Ensure this ID exists for deletion assertThat(userRepo.existsById(2L)).isFalse(); // Check that it no longer exists**

**}**

**}**

**Step 6: Run Your Tests**

1. **Run Test Class:**

o **Right-click on the SpringbootFirstAppApplicationTests.java file.** o **Select Run As → JUnit Test.**

1. **View Results:**

o **Check the JUnit view to see if your tests passed or failed. If they fail, check the error messages to debug.**

**17. CRUD Operations on document using Mongo DB**

**Step 1: Set Up MongoDB**

**1. Install MongoDB (if you haven’t already):**

**3.**

**Show Collections**

**:**

o

To confirm the collection was

created, list all collections

**Step 3: Insert Data**

**1.**

**Insert a Single Document**

**:**

o

**Use the insert() method**

**1.**

**Insert Multiple Documents**

**:**

o

**Use the insertMany() method:**



o **Download from the** [**MongoDB website.**](https://www.mongodb.com/try/download/community)

**Follow the installation instructions for your operating system.**

1. **Start MongoDB**

**Open a terminal (or Command Prompt).**

* A white background with black and white clouds

  Description automatically generated with medium confidence**Start the MongoDB server with:**
* **Leave this terminal open to keep the server running.**

1. **. Open a New Terminal**

**In a new terminal, start the MongoDB shell by typing**

A white background with black and white clouds

Description automatically generated with medium confidence

**Step 2: Create a Database and Collection**

**Switch to a database**

**If you want to create a new database (e.g., school), type**

A white background with black and white clouds

Description automatically generated with medium confidence **This command creates the database if it doesn't already exist.**

**2. Create the Collection**

**Create a collection named "student"**

A white background with text and numbers

Description automatically generated

**Step 4: View Data**

**Retrieve All Documents**

**To view all documents in the "student" collections**

A screenshot of a computer

Description automatically generated

**Step 5: Update Data**

**Update a Document**

**Use the update() method to modify a document:**

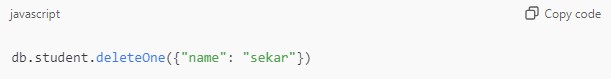
A white background with text

Description automatically generated

**Step 6 :Delete Data**

**Delete a Single Document**

**Use the deleteOne() method:**



**18. Perform CRUD Operations on MongoDB through REST API using Spring Boot Starter Data MongoDB**

**Step 1: Create a Spring Boot Project**

1. **Use Spring Initializr:** 
   * **Go to** [**Spring Initializr.**](https://start.spring.io/)
   * **Choose the following settings:** 
     + **Project: Maven Project**
     + **Language: Java**
     + **Spring Boot: Select the latest stable version.**
   * **Project Metadata:** 
     + **Group: com.example**
     + **Artifact: bookstore** o **Dependencies: Add the following dependencies:**
     + **Spring Web**
     + **Spring Data MongoDB**
     + **Lombok**
     + **Spring Boot DevTools** o **Click Generate to download the project.**
2. **Extract and Import the Project:** 
   * **Extract the downloaded zip file.**
   * **Open your IDE (e.g., IntelliJ or Eclipse) and import the project as a Maven project. Step 2: Create Package Structure**

**1. Create Packages:**

o **Inside the src/main/java/com/example/bookstore directory, create the following packages:**

* **entity**
* **repository**
* **controller**

**Step 3: Create the Book Entity**

**1. Create Book.java:**

o **Inside the entity package, create a file named Book.java and add the following code:**

package com.example.bookstore.entity; import lombok.AllArgsConstructor; import lombok.Data; import lombok.NoArgsConstructor; import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Document(collection = "Book") public class Book {

@Id

private int id;

private String bookName;

private String authorName;

}

**Step 4: Create the Repository**

1. **Create BookRepo.java:**

o **Inside the repository package, create a file named BookRepo.java and add the following code:**

package com.example.bookstore.repository;

import com.example.bookstore.entity.Book; import org.springframework.data.mongodb.repository.MongoRepository;

public interface BookRepo extends MongoRepository<Book, Integer> { }

**Step 5: Create the Controller**

**1. Create BookController.java:**

* + **Inside the controller package, create a file named BookController.java and package com.example.bookstore.controller;**

o

* + **import com.example.bookstore.entity.Book;** o **import com.example.bookstore.repository.BookRepo;** o **import org.springframework.beans.factory.annotation.Autowired;** o **import org.springframework.web.bind.annotation.\*;** o
  + **import java.util.List;**

o

* + **@RestController** o **public class BookController {** o  **@Autowired** o  **private BookRepo repo;**

o

* + **@PostMapping("/addBook")** o  **public String saveBook(@RequestBody Book book) {** o  **repo.save(book);** o  **return "Added Successfully";** o  **}**

o

* + **@GetMapping("/findAllBooks")** o  **public List<Book> getBooks() {**
  + **return repo.findAll();** o  **}**

o

* + **@DeleteMapping("/delete/{id}")** o  **public String deleteBook(@PathVariable int id) {** o  **repo.deleteById(id);** o  **return "Deleted Successfully";** o  **}**
  + **}**

**Step 6: Configure Application Properties**

**1. Edit application.properties:**

* + **Open src/main/resources/application.properties and add the following lines:**

server.port=8989 spring.data.mongodb.host=localhost spring.data.mongodb.port=27017

spring.data.mongodb.database=jss

**Step 7: Set Up MongoDB**

* 1. **Open MongoDB Compass:** o **Launch MongoDB Compass.**
  2. **Create a Database:** o **Create a new database named BookStore.**
  3. **Create a Collection:** o **Inside the BookStore database, create a collection named Book.**

**Step 8: Run the Application**

**1. Run Your Spring Boot Application:**

* + **In your IDE, run the main application class (e.g., BookstoreApplication.java).**

**19. Securing REST APIs with Spring Security**

**Step 1: Create a New Spring Boot Project**

1. **Use Spring Initializr**:

* + Go to [Spring Initializr.](https://start.spring.io/)
  + Choose the following settings:
    - **Project**: Maven Project
    - **Language**: Java
    - **Spring Boot**: Select the latest stable version.
  + **Project Metadata**:
    - **Group**: com.example
    - **Artifact**: spring-basic-security o **Dependencies**: Add the following dependencies:
    - Spring Web
    - Spring Security
    - Spring Boot DevTools o Click **Generate** to download the project.

1. **Extract and Import the Project**:
   * Extract the downloaded zip file.
   * Open your IDE (e.g., IntelliJ or Eclipse) and import the project as a Maven project.

**Step 2: Add Spring Security Dependency**

1. **Open pom.xml**:

o In the pom.xml file, ensure the Spring Security dependency is included:

<dependency>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-security</artifactId> </dependency>

**Step 3: Create the Main Application Class**

**1. Create SpringBasicSecurityApplication.java:**

o **Inside the com.example.security package, create a file named**

**SpringBasicSecurityApplication.java and add the following code:**

**package com.example.security;**

**import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;**

**@SpringBootApplication public class SpringBasicSecurityApplication { public static void main(String[] args) {**

**SpringApplication.run(SpringBasicSecurityApplication.class, args);**

**}**

**}**

**Step 4: Create a Security Controller 1. Create SecurityController.java:**

o **Inside the com.example.security package, create a file named SecurityController.java and add the following code:**

**package com.example.security;**

**import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController;**

**@RestController public class SecurityController {**

**@GetMapping("/") public String welcome() { return "<h1>Welcome to Spring Boot Security</h1>";**

**}**

**}**

**Step 5: Configure Application Properties**

**1. Edit application.properties:**

o **Open src/main/resources/application.properties and add the following lines:**

**spring.security.user.name=niranjan spring.security.user.password=murthy server.port=8090**

**Step 6: Run the Application**

1. Run Your Spring Boot Application: o In your IDE, run the main application class (SpringBasicSecurityApplication.java).

**Step 7: Access the Secured Endpoint**

1. **Open a Web Browser or Postman:** 
   * Navigate to http://localhost:8090/.
2. **Authentication Prompt:** 
   * You should see a login prompt. Enter the username and password configured in the application.properties:
     + Username: niranjan
     + Password: murthy
3. **View the Welcome Message:** o After successful authentication, you should see the message: - Welcome to Spring Boot Security

**Step 8: Default Password Behavior**

• **If you don’t set a username and password in application.properties, Spring Security will generate a random password, which will be displayed in the console logs when you start the application.**

**20. Build simple page application like shopping cart using ReactJS.**

**Step 1: Set Up the React Project 1. Create a New React App:**

A screenshot of a computer

Description automatically generated **Step**

**2: Create the Component Files**

1. **Navigate to the src Folder**: o Inside the src directory, create the following files:

* App.js
* Header.js
* Product.js
* CartList.js

**Step 3: Implement App.js**

1. **Open App.js** and add the following code:

import Header from "./Header"; import Products from "./Product"; import { useState } from "react";

import CartList from "./CartList";

function App() {

const [product, setProduct] = useState([

{

url: 'imgs/lenovo.png', name: 'Lenovo Ideapad Slim 3',

price: 57000

},

{

url: 'imgs/watch.png', name: 'Fastrack W98',

price: 1500

},

]);

const [cart, setCart] = useState([]);

const [showCart, setShowCart] = useState(false);

const addToCart = (data) => {

setCart([...cart, { ...data, quantity: 1 }]);

};

const handleShow = (value) => {

setShowCart(value);

};

return ( <div>

<Header count={cart.length} handleShow={handleShow} /> {showCart ?

<CartList cart={cart} /> :

<Products product={product} addToCart={addToCart} />

}

</div>

);

}

export default App;export default App;

**Step 4: Implement Product.js**

**1. Open Product.js and add the following code:**

import React from 'react';

export default function Products({ product, addToCart }) { return (

<div className='flex'>

{product.map((productItem, productIndex) => { return (

<div key={productIndex}>

<img src={productItem.url} width="20%" alt="" />

<p>{productItem.name}</p>

<p>Rs.{productItem.price}</p>

<button onClick={() => addToCart(productItem)}>Add to Cart</button>

</div>

);

})}

</div>

);

}

**Step 5: Implement CartList.js**

1. **Open CartList.js** and add the following code:

import React, { useState, useEffect } from 'react';

function CartList({ cart }) {

const [CART, setCART] = useState([]);

useEffect(() => { setCART(cart);

}, [cart]);

return ( <div>

{CART?.map((cartItem, cartIndex) => { return (

<div key={cartIndex}>

<img src={cartItem.url} width={60} />

<span>{cartItem.name}</span> <button onClick={() => {

const \_CART = CART.map((item, index) => {

return cartIndex === index ? { ...item, quantity: item.quantity > 0 ? item.quantity - 1 : 0 } : item;

});

setCART(\_CART);

}}>-</button>

<span>{cartItem.quantity}</span>

<button onClick={() => {

const \_CART = CART.map((item, index) => {

return cartIndex === index ? { ...item, quantity: item.quantity + 1 } : item;

});

setCART(\_CART);

}}>+</button>

<span> Rs.{cartItem.price \* cartItem.quantity}</span> </div>

);

})}

<p>Total =

<span>{CART.map(item => item.price \* item.quantity).reduce((total, value) => total + value, 0)}</span>

</p>

</div>

);

}

export default CartList;

**Step 6: Implement Header.js**

1. **Open Header.js** and add the following code:

import React from 'react';

export default function Header(props) { return ( <div>

<div onClick={() => props.handleShow(false)}>Shopping Cart</div>

<div onClick={() => props.handleShow(true)}>

Cart <sup>{props.count}</sup>

</div>

</div>

);

}

**Step 7: Add Images**

1. **Add Images to Your Project**:

* Create an imgs folder in the public directory.
* Add images named lenovo.png and watch.png to the imgs folder. **Step 8: Style Your Application**

1. **Optional Styling**:

o You can add CSS styles in App.css to improve the layout. Here’s a simple example:

.flex { display: flex; flex-wrap: wrap;

}

div { margin: 10px; padding: 10px; border: 1px solid #ccc; text-align: center;

}

**Step 9: Run Your Application**

1. **Run the Application**:

* Make sure your development server is running (npm start).
* Open your browser and go to http://localhost:3000.